SharePoint security accounts

Pav Cherny

When using SharePoint security accounts, there is a high risk of creating weak system configurations that can expose an entire SharePoint environment. To help you deploy and secure SharePoint server farms correctly, Microsoft has published extensive information and detailed guidelines.

The Office SharePoint Server Security guide (available at *go.microsoft.com/fwlink/?LinkId=84739*), for example, comprises more than 300 pages about planning and implementing site and content hierarchies, authentication methods, security roles, administrator and service accounts, and many other security issues. The Windows SharePoint Services Security Account Requirements worksheet (see *go.microsoft.com/fwlink?LinkId=92930&clcid=0x409*) also provides essential information regarding security account configurations. If security is important to you, you definitely want to make sure to follow this worksheet.

Even with extensive documentation, configuring security accounts can be a difficult task. In fact, the default settings of single-server installations deviate from the worksheet's recommendations, and certain components, such as the E-mail Integration Web Service included in Windows SharePoint Services (WSS) 3.0, require elevated permissions on the server, which not only strays from the Microsoft security recommendations but is in direct conflict with security best practices and plain common sense. The SharePoint 3.0 Central Administration tool happily applies critical security account configurations without warnings, the Microsoft Baseline Security Analyzer (MBSA) does not detect the resulting weaknesses, and so it remains a challenge to secure a SharePoint server farm—and to keep it secure.

In this column, I put SharePoint security accounts under a microscope to show you how a weak configuration can give an attacker full control over all site collections and sites. This is a somewhat sensitive topic. On the one hand, I want to help you recognise the security challenges that surround SharePoint server configurations. After all, you must understand both the strengths and the weaknesses of your SharePoint environment if you want to secure it effectively.

On the other hand, I don't want to aid malicious people. For this reason, I am not providing any worksheets or custom tools with this column and I'll constrain source code discussions to basic topics that should be familiar to every professional ASP.NET developer. The source code snippets covered in this column should help you detect vulnerabilities, but not help anyone exploit them. Even with limited programming skills, you should be able to use these snippets to create custom ASP.NET pages if you have Microsoft Office SharePoint Designer 2007.

A trial version of Microsoft Office SharePoint Designer 2007 is available at *go.microsoft.com/fwlink/?LinkId=130166*. I invite you to configure a test lab according to your own preferences, secure it as best as you can, and then run the code snippets for verification

SharePoint Resources

- **SharePoint Products and Technologies Web Site**
  microsoft.com/sharepoint
- **Windows SharePoint Services TechCenter**
  technet.microsoft.com/windowsserver/sharepoint
- **Windows SharePoint Services Developer Center**
  msdn.microsoft.com/sharepoint
- **Microsoft SharePoint Products and Technologies Team Blog**
  blogs.msdn.com/sharepoint

purposes. Let's see how secure your SharePoint sites are!

Application pools and security accounts

Security accounts are at the very core of the SharePoint request-processing model. They define the security context of the IIS worker processes that run the SharePoint Web applications. When you create a SharePoint Web application, you must specify, among other things, an application pool with associated security account credentials and a SharePoint database with an associated authentication method. If you use Windows authentication (recommended), SharePoint automatically grants the specified security account dbOwner permissions on the content database so that the IIS worker process running the SharePoint Web application can gain access to the site collections and sites hosted in this database. Otherwise, you must provide explicit SQL Server credentials.

In any case, SharePoint site collections and sites are virtual constructs. Physically, they correspond to database records. If you know the account name and password to establish a direct SQL Server connection to the content database, you can gain full access to all its site collections and site data regardless of permissions and access controls defined at the SharePoint level. SharePoint cannot block you because you are establishing a direct connection to the database server, as illustrated in **Figure 1**. The security account is therefore a prime target for an attack.

If you know the account name and password, you gain full access to all site collections and site data, regardless of permissions.

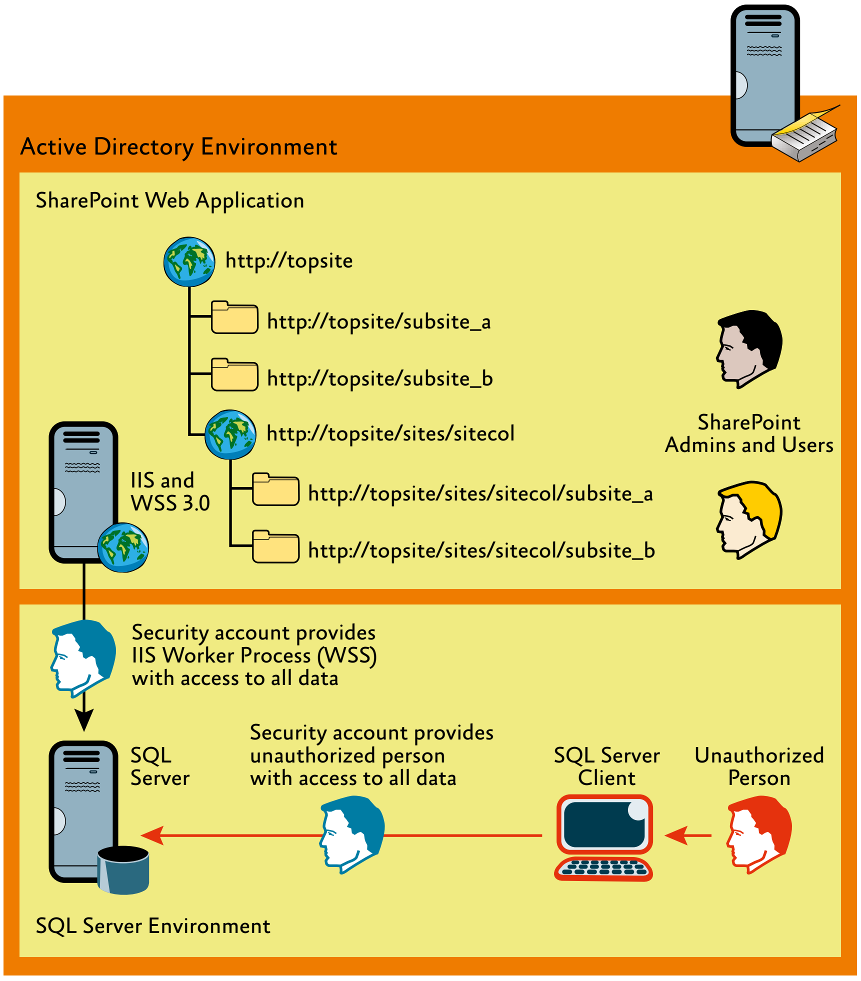

Active Directory Environment
SharePoint Web Application
http://topsite
http://topsite/subsite_a
http://topsite/subsite_b
http://topsite/sites/sitecol
http://topsite/sites/sitecol/subsite_a
http://topsite/sites/sitecol/subsite_b
IIS and WSS 3.0
SharePoint Admins and Users
Security account provides IIS Worker Process (WSS) with access to all data
SQL Server
Security account provides unauthorized person with access to all data
SQL Server Client
Unauthorized Person
SQL Server Environment

Figure 1 **Bypassing SharePoint site collections and sites to access data**

To mitigate security risks, Microsoft recommends that you configure separate applications pools (and security accounts) for site collections with authenticated and anonymous content and to isolate applications that store passwords or where users have great liberty to create and administer sites and to collaborate on content. By following this configuration advice, the underlying idea is that an attacker who gains control over one application pool will not then implicitly have universal access to all data hosted in the SharePoint farm. SharePoint site collections and sites in other databases are still out of reach, provided you use separate security accounts for their associated Web applications.

Microsoft first introduced the concept of worker process isolation based on applications pools with IIS 6.0 and states that IIS has not suffered a single critical security vulnerability ever since. This is very reassuring, so be sure to take advantage of application pools in SharePoint farms. Keep in mind, however, that IIS Web sites and SharePoint Web applications are not synonymous. While you can isolate IIS Web sites, you cannot isolate SharePoint Web applications from each other.

Genuine isolation exists only if Web sites do not share resources, yet SharePoint Web applications always have resources in common, such as the farm's configuration database. As illustrated in **Figure 2**, gaining control over a SharePoint security account implies the ability to access the SharePoint configuration database. That

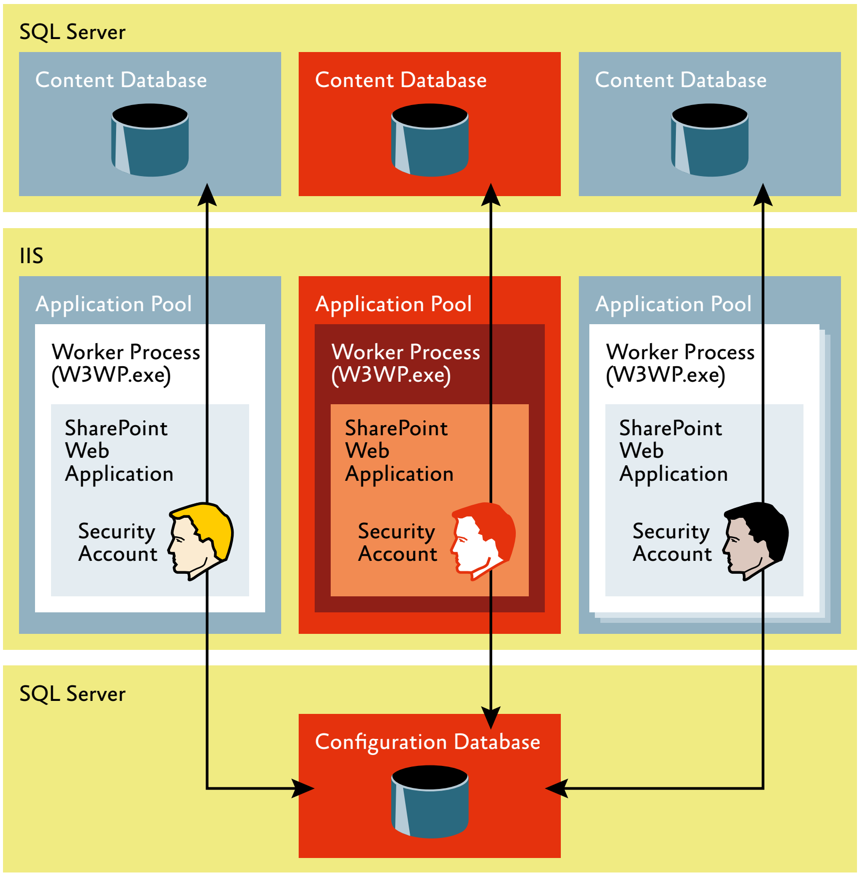


Figure 2 **Relationship between SharePoint Web applications, configuration database, and content databases**

access capability should make you feel uncomfortable if you deployed your SharePoint farm without giving proper thought to the protection of your security accounts, especially if you are hosting site collections and sites from different internal or external customers in a shared environment.

Running custom code on SharePoint Servers

Out of the box, SharePoint doesn't disclose security account information; it takes malicious code to discover the details. As we all know, security vulnerabilities can enable attackers to upload malicious code, but sometimes intrusion is even easier.

In the simplest case, an attacker might be able to log on to a SharePoint server locally or via Terminal Server and copy malicious code into the %COMMONPROGRAMFILES%\Microsoft Shared\Web Server Extensions\12\TEMPLATE\Layouts folder. Note that SharePoint includes this folder as a virtualised subfolder in every SharePoint site.

In another scenario, a SharePoint administrator might unknowingly introduce malicious code by deploying a custom SharePoint solution from a questionable source without proper testing and code verification. Inline ASP.NET code embedded in master and content pages is also cause for concern. By default, SharePoint does not process server-side scripts, but anyone with write-access to a SharePoint Web application's web.config file can change the rules for processing server-side scripts. You need only add a PageParserPath entry to the web.config file's <PageParserPaths> section. How exactly does the PageParserPath entry work?

Let's suppose a developer who is using SharePoint calls you and complains about an error message he gets while developing a custom ASP.NET page that states "Code blocks are not allowed in this file." You search the Internet and find the solution in a newsgroup or blog site:

```
<PageParserPath VirtualPath="/*"
CompilationMode="Always"
    AllowServerSideScript="true"
IncludeSubFolders="true" />
```

Perhaps you ignore the security warnings or perhaps the security implications aren't even mentioned. No matter, you add this line to your web.config file and now everybody is happy because you solved the problem.

Unwittingly, however, you also just opened an avenue to run any custom code in ASP.NET pages with full trust. If an attacker now uploads a malicious ASP.NET page, the SharePoint environment is in jeopardy. As indicated in **Figure 3**, it is irrelevant where in a site collection hierarchy an attacker has the permission to upload a page – it can be an innocent-looking small team site. The attack always affects the Web application and possibly the en-

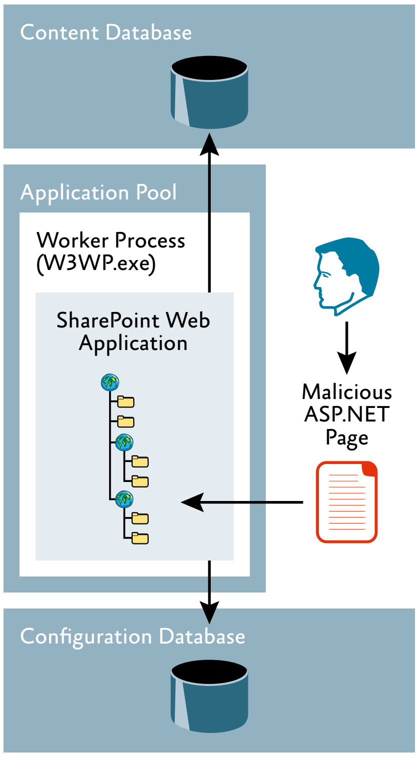


Figure 3 **Enabling inline ASP.NET code can compromise a SharePoint Web application**

tire server farm because both content database and configuration database are accessible.

About Dr. Jekyll and Mr. Hyde

So how does an attacker gain access to content and configuration databases without explicitly knowing the security account credentials? It's actually relatively straightforward. The IIS worker process that runs the SharePoint Web application impersonates the SharePoint user and uses the resulting thread token for access checks. For example, Dr. Jekyll can access all those SharePoint resources that his security token is permitted to access. But the SharePoint Web application also has the process token of the IIS worker process, which is the security token of the SharePoint security account.

It is Mr. Hyde who shows up when you revert impersonation by calling the static WindowsIdentity.Impersonate method, and passing in a zero pointer, as illustrated in **Figure 4**. Dr. Jekyll has no direct access to the databases, but Mr. Hyde does. The road is clear for SQL Server connections and T-SQL queries.

One way to deliver security credentials into the wrong hands is to grant application pool accounts access to the IIS metabase.

Security accounts and process isolation

Application pools and security accounts cannot help you protect site collections and sites placed in a Web application configured to run unverified code. Their purpose is to mitigate, by means of process isolation, the impact of an exploit on one site that allows an attacker to inject code onto the server to attack other sites. Process isolation can help to achieve this goal, but it requires you to place the other sites in separate Web applications that run in application pools with different security accounts.

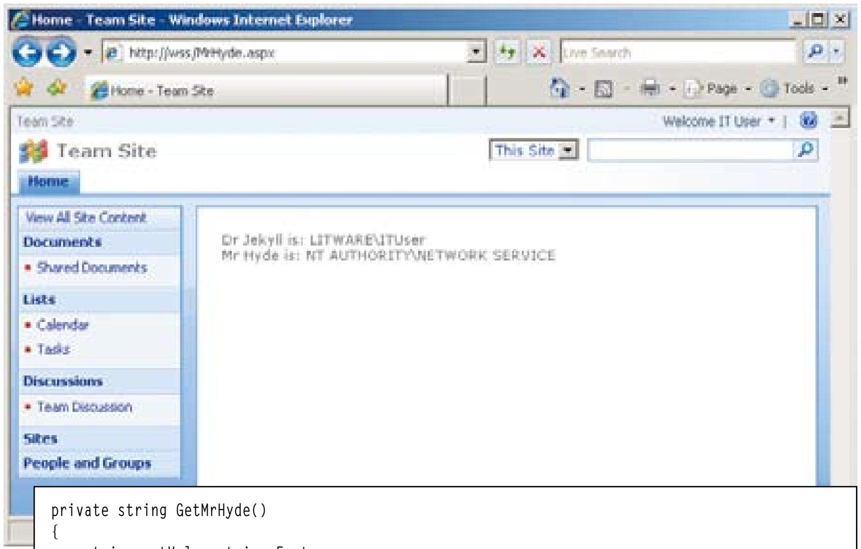
You must adequately protect the account credentials, otherwise the configuration effort is pointless. One easy way to deliver these sensitive security credentials straight into the wrong hands is to grant application pool accounts access to the IIS metabase, which is required to run the Directory Management Service—a part of the E-mail Integration Web service. If the application pool account has metabase access, an attacker can revert impersonation and then retrieve all the accounts and passwords in clear text, as illustrated in **Figure 5**. The entire server farm is lost because the attacker can now bypass process isolation by running malicious code under any of these security accounts and establishing SQL Server connections to all content databases.

If you're interested in a Directory Management Service solution that doesn't require metabase access, you should look up my September 2008 column, "SharePoint Directory Integration," at technet.microsoft.com/magazine/cc742803.

Security accounts in the configuration database

If you follow the rule not to grant application pool accounts administrative permissions or even so much as read access to the IIS metabase on your SharePoint servers, the code in **Figure 5** only yields an Access is denied message. But security account information is also available in the configuration database and Mr. Hyde has access to this database, as explained earlier.

You can't deny SharePoint security accounts access to the configuration database nor can you deny access to the registry key that stores the correspond-



```
Dr Jekyll is: LITWARE\ITUser
Mr Hyde is: NT AUTHORITY\NETWORK SERVICE
```

```
private string GetMrHyde()
{
    string retVal = string.Empty;
    retVal = "Dr Jekyll is: " + WindowsIdentity.GetCurrent().Name + "<br>";

    WindowsImpersonationContext impCtx = WindowsIdentity.Impersonate(IntPtr.Zero);

    retVal += "Mr Hyde is: " + WindowsIdentity.GetCurrent().Name + "<br>";

    impCtx.Undo();
    return retVal;
}
```

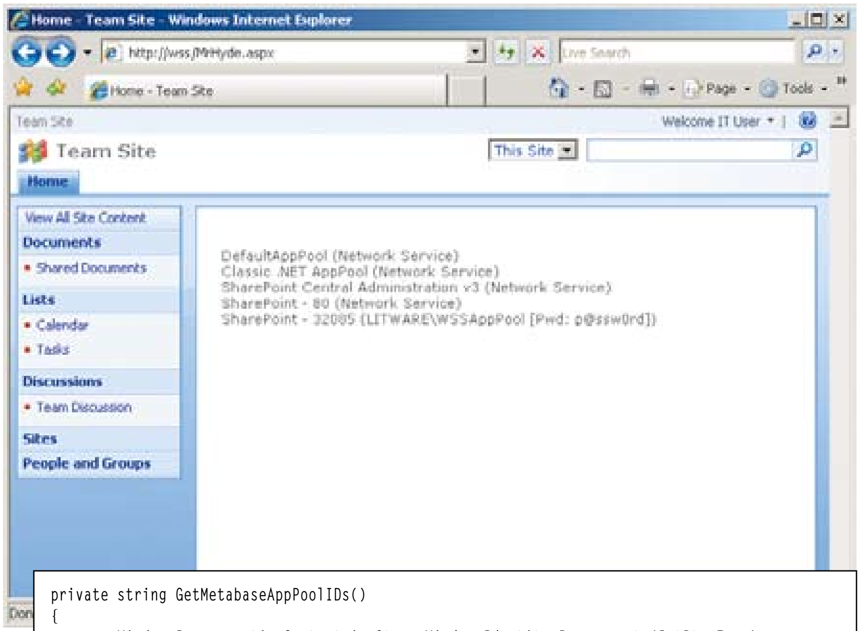
Figure 4 **SharePoint Web applications have two security contexts**

ing SQL Server connection string. The connection string might not immediately work if you use SQL Server 2005 Express, but you can derive the correct data source information from the current SharePoint site collection (SPContext.Current.Site.ContentDatabase.DatabaseConnectionString) and the name of the configuration database corresponds to the name of the local farm (SPFarm.Local.Name).

Unfortunately, these little hurdles don't stop an attack. Regardless of whether you use SQL Server or SQL Server Express, Mr. Hyde can retrieve the information displayed in **Figure 6**. Note that the password is encrypted, however, so the attack has not yet fully succeeded.

But even without decrypting passwords, application pools that use the same security account are already recognizable. For example, in **Figure 6**, the application pools SharePoint Central Administration v3 and SharePoint—80 use the Network Service account and if SharePoint—80 happens to be the compromised Web application, then SharePoint Central Administration v3 is compromised as well. The corresponding security account is the Central Administration account, which has elevated permissions on the SharePoint server.

It should not be used for standard Web application pools, yet the SharePoint Products and Technologies Configuration Wizard applies this configuration in a single-server installation by default. Therefore, it's very important that you review and, if necessary, change the security account configuration in your SharePoint environment. More information on this topic is outlined in detail in the Microsoft Knowledge Base article "How to Change Service Accounts and Service Account Passwords in SharePoint Server 2007 and in Windows SharePoint Services 3.0" (see support.microsoft.com/kb/934838).



```
DefaultAppPool (Network Service)
Classic .NET AppPool (Network Service)
SharePoint Central Administration v3 (Network Service)
SharePoint - 80 (Network Service)
SharePoint - 32095 (LITWARE\WSSAppPool [Pwd: p@ssw0rd])
```

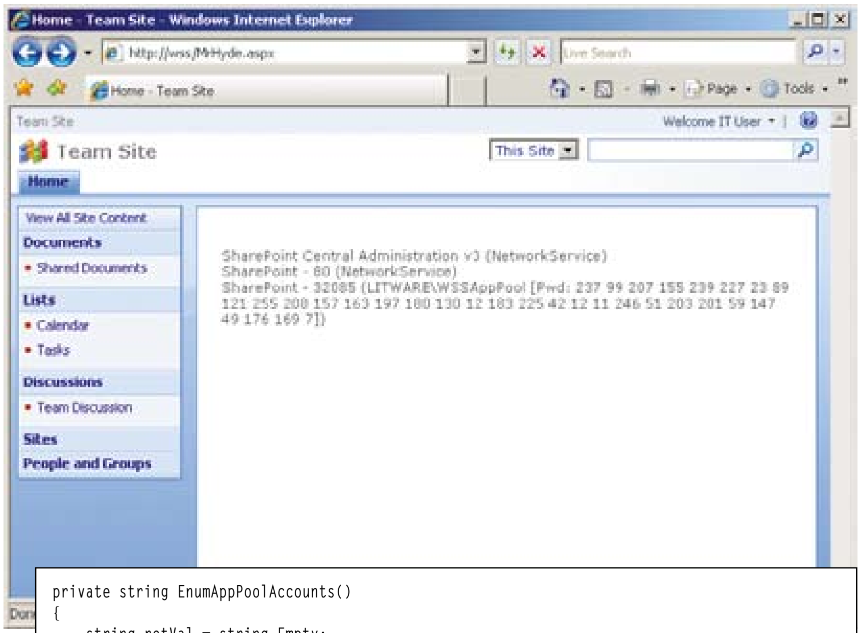
```
private string GetMetabaseAppPoolIDs()
{
        WindowsImpersonationContext impCtx = WindowsIdentity.Impersonate(IntPtr.Zero);
        string retVal = string.Empty;
        try
        {
            string metabasePath = "IIS://localhost/w3svc/AppPools";
            DirectoryEntry appPools = new DirectoryEntry(metabasePath);
            foreach (DirectoryEntry appPool in appPools.Children)
            {
            switch (int.Parse(appPool.Properties["AppPoolIdentityType"].Value.ToString()))
            {
                case 0: // Local System
                    retVal += "<br>" + appPool.Name
                        + " (Local System)";
                    break;
                case 1: // Local Service
                    retVal += "<br>" + appPool.Name
                        + " (Local Service)";
                    break;
                case 2: // Network Service
                     retVal += "<br>" + appPool.Name
                        + " (Network Service)";
                     break;
                case 3: // Custom
                    retVal += "<br>" + appPool.Name
                        + " (" + appPool.Properties["WAMUserName"].Value
                        + " [Pwd: " + appPool.Properties["WAMUserPass"].Value
                        + "])";
                    break;
               }
            }
         }
        catch (Exception ee)
         {
        retVal = "Metabase " + ee.Message;
         }

        impCtx.Undo();
}
```

Figure 5 **Retrieving security account information from the IIS metabase**

Security accounts and credential keys

So what's the big deal about the Central Administration account? Most importantly, unlike standard application pool accounts, the Central Administration account has access to the registry location that stores the credential key to decrypt the security account passwords.

Figure 7 shows this parameter and its default security settings. As you can see, local Administrators, the WSS_RESTRICTED_WPG group (which contains the Central Administration



```
SharePoint Central Administration v3 (NetworkService)
SharePoint - 80 (NetworkService)
SharePoint - 32085 (LITWARE\WSSAppPool [Pwd: 237 99 207 155 239 227 23 89
121 255 200 157 163 197 100 130 12 183 225 42 12 11 246 51 203 201 59 147
49 176 169 7])
```

```
private string EnumAppPoolAccounts()
{
    string retVal = string.Empty;
    try
    {
       WindowsImpersonationContext impCtx = WindowsIdentity.Impersonate(IntPtr.Zero);

       string regConfigDB =
          @"SOFTWARE\Microsoft\Shared Tools\Web Server Extensions\12.0\Secure\ConfigDB";
       RegistryKey keyConfigDB = Registry.LocalMachine.OpenSubKey(regConfigDB);

       string ConfigDB = (string)keyConfigDB.GetValue("dsn");

       SqlConnection sqlConn = new SqlConnection(ConfigDB);
       sqlConn.Open();

       SqlCommand sqlCmd = new SqlCommand("SELECT Name, Properties FROM Objects"
          + " WHERE ClassId = 'B8369089-08AD-4978-B1CB-C597B5E90F64'", sqlConn);
       sqlCmd.CommandType = System.Data.CommandType.Text;
       SqlDataReader sqlReader = sqlCmd.ExecuteReader();

          while (sqlReader.Read())
          {
             retVal += "<br>" + sqlReader.GetString(0);
             string appPoolXML = sqlReader.GetString(1);
             if (!string.IsNullOrEmpty(appPoolXML))
             {

                 XmlDocument xmlDoc = new XmlDocument();
                 xmlDoc.LoadXml(appPoolXML);

                 XmlElement root = xmlDoc.DocumentElement;
                 XmlNode ndType = root.SelectSingleNode("/object/fld[@name=
                   'm_IdentityType']");
                 if (ndType != null && ndType.InnerText.ToLower() != "specificuser")
                 {
                     retVal += " (" + ndType.InnerText + ")";
                 }
                 else
                 {
                     retVal += " ("
                         + root.SelectSingleNode("/object/sFld[@name='m_Username']").InnerText
                         + " [Pwd: "
                         + root.SelectSingleNode("/object/fld[@name='m_Password']").InnerText
                         + "])";
                 }
             }
          }
          sqlReader.Close();
          sqlConn.Close();
          impCtx.Undo();
    }
    catch (Exception ee)
    {
          retVal = ee.Message;
    }
    return retVal;
}
```

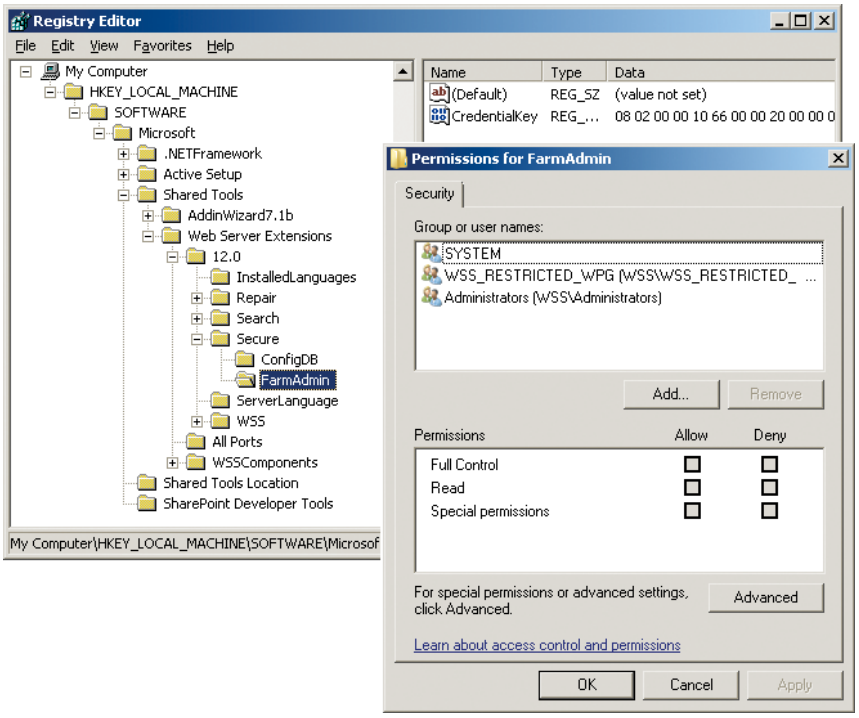
Figure 6 **Getting security account information from the configuration metabase**

account), and the SYSTEM account have access to this key and this implies that your SharePoint Web applications should not use accounts with local administrator permissions, the Central Administration account, or the SYSTEM account. SharePoint Web applications should not be able to access the credential key.

Unfortunately, however, this is no guarantee that a skilled attacker cannot determine the CredentialKey or security account passwords, such as through SYSTEM token hijacking, password cracking, or simply by placing malicious code in master pages or content pages to export the credential key to an unprotected location and then waiting for a user with local administrator permissions to access the site. As you can see, it's important that you don't allow unverified code on your servers.

SYSTEM token hijacking deserves some more detailed explanation because you can prevent this form of attack if you avoid using the built-in system accounts, such as Network Service, for your SharePoint Web applications. In fact, Cesar Cerrudo, founder and CEO of Argeniss, discovered this vulnerability, and he demonstrated the exploit at the HITBSecConf2008 Deep Knowledge Security Conference in Dubai, United Arab Emirates. Cesar showed how an ASP.NET Web application running under the Network Service account can inject a DLL into the Remote Procedure Call (RPC) service and then hijack the security token of a thread in the RPC service that runs at SYSTEM privilege level.

After this, an attacker then only needs to pass the hijacked SYSTEM security token to the WindowsIdentity.Impersonate method to gain access to the CredentialKey registry parameter and other protected resources. Microsoft confirmed the vulnerability, so you should avoid using the Network Service account for your SharePoint Web applications.



```
My Computer
  HKEY_LOCAL_MACHINE
    SOFTWARE
      Microsoft
        .NETFramework
        Active Setup
        Shared Tools
          AddinWizard7.1b
          Web Server Extensions
            12.0
              InstalledLanguages
              Repair
              Search
              Secure
                ConfigDB
                FarmAdmin
              ServerLanguage
              WSS
            All Ports
          WSSComponents
        Shared Tools Location
        SharePoint Developer Tools
My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsof
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
CredentialKey	REG_...	08 02 00 00 10 66 00 00 20 00 00 0

Group or user names:

- SYSTEM
- WSS_RESTRICTED_WPG (WSS\WSS_RESTRICTED_ ...
- Administrators (WSS\Administrators)

Permissions	Allow	Deny
Full Control	☐	☐
Read	☐	☐
Special permissions	☐	☐

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

Figure 7 **Permission assignments to access the FarmAdmin registry key**

Don't break the law

The 10 Immutable Laws of Security were published by the Microsoft Security Response Center a long time ago (go to technet.microsoft.com/library/cc722487), and they still apply today. Jesper M. Johansson recently wrote a three-part series called "Revisiting the 10 Immutable Laws of Security," at technet.microsoft.com/magazine/cc895640). You should keep these laws in mind when designing your SharePoint server farms, and you should also follow the SharePoint security guidelines and worksheets to apply reliable security account configurations.

In a nutshell: use strong passwords, don't grant security accounts elevated permissions on your SharePoint servers, change passwords frequently (including the farm credentials), and keep in mind that there is no absolute isolation between SharePoint Web applications that use common resources, just as there is no absolute computer security. Moreover, don't change the server code processing rules, keep unverified assemblies away from your servers, and follow the Windows SharePoint Services Security Account Requirements worksheet in your security account configuration, and you might be able to consider your SharePoint environment reasonably secure.

However, if strict separation of site content is a requirement for your organisation, I recommend that you host the corresponding site collections in separate server farms, possibly in separate Active Directory forests and SQL Server environments. ■

For more information visit:
http://technet.microsoft.com/en-gb/office/sharepointserver/default.aspx

Pav Cherny *is an IT expert and author specialising in Microsoft technologies for collaboration and unified communication. His publications include white papers, product manuals, and books with a focus on IT operations and system administration. Pav is President of Biblioso Corporation, a company that specialises in managed documentation and localisation services.*